Application Development Standards

The Application Development Standards and Procedures deliverable consist of the standards, rules, and guidelines to be followed during the application development process for programming and documenting programs. This document is not meant to be a training manual. Rather, it is a reference manual for the standards set by the development architecture. Use the Application Development Standards and Procedures to provide a consistent way of designing, documenting, programming, etc., over the different areas of work, such as user interface design and data design. Quality reviews may be drawn from this deliverable.

The application development standards and procedures may contain the following:

- Programming Standards
- Performance Design Guidelines (e.g., 2MB limits on transaction size during peak hours)
- File/Directory naming standards
- Application Program Interface (API) description and use
- Procedures for using the development architecture and its operations support components (e.g., checking in and out code, when backups occur and how to retrieve archived files, compiling programs)

I. IPT Name:		
II. Deliverable Name: Application Development Standards		**Date Completed:**
III. Contact Information		
	Name	Channel Unit
IPT Sponsor		
Channel Task Manager		
CIO Task Manager		
Contractor Task Manager		
IV. Task Order Number:		

Description

This sample provides an example of the naming conventions, the directory structures, and C/C++ programming standards.

For every programming environment, there will be different standards and guidelines that are specific for the environment. There are, however, a couple of sections that exist in almost all programming standards:

Directory Structures
Naming Conventions

Other, more language-specific sections are:

Variable and Parameter Declarations (Procedural Languages)
Class Structure (Object-Oriented Languages)
Query Standards (Query Languages)

and so forth.

Application Development Standards and Procedures

Table of Contents

Introduction

The objective of this document is to define a number of standards and procedures in the area of Application Development for the XYZ project. It defines the following standards:

- Naming Conventions
- Directory Structure
- C / C++ Programming Standard

Naming Conventions

Each file will consist of an eight character name. The name will be broken up into three main sections - a program identifier, a descriptive name identifier, a numeric identifier and an extension:

yzzzzznn.ext	
y	Application type identifier
zzzzz	Descriptive name
nn	Unique numeric identifier

The application type will designate what type of file it is (help file, visual basic, c++, etc.)
The descriptive name will be a five character (or less) name describing the contents of the file. These will vary based on application type.
The numeric identifier will uniquely identify a file within the program type / descriptive name designation.
The extension is usually imposed by the application where the file is being created / used.

Application Type	Application Type Identifier	Extension	Example
Dynamic Link Libraries	D	.DLL	DMAINT03.DLL
Help File	H	.HLP	HMAINT11.HLP
Visual Basic	V	.BAS or .VBX	VENTRY01.BAS
C	C	.C or .MAK or .EXE	CEXTRN02.C
C++	P	.C or .MAK or .EXE	PINVCE01.EXE

Directory Structure

All of the application type files will reside on the network and will be checked out with version control software. The server directory structure and its contents are as follows:

```
G:\
\Sales\                 - Project Root directory
\C            - C files
\CPLUS        - C++ files
\DLL          - DLL files
\Help         - Help files
\VB           - Visual Basic files
\VBCom        - Visual Basic common modules
\Team         - Each team member's files (first initial, last name)
\JDoe
\SSmith
.
.
.
\WTaylor
\Custserv     - Next project directory
\C
\DLL
etc.
```

C/C++ Programming Standard

C and C++ Sources

Debugging

RulePROG1: Prefer compilation errors to execution errors. Use #error if necessary.

RulePROG2: Enforce the programs by verifying the assertions.
This allows you to obtain the development version from which the errors originate.

RulePROG3: Construct types for all objects.
Do not use *float* for a price.

RulePROG4: Each variable should be initialized with a value before it is used.
Initialize all variables, even if it is zero.

RulePROG5: Always allocate the exact number of 8-bits in a buffer.
Do not add a few 8-bits to reduce the risk of error. This allows you to rapidly detect errors that result from passing the buffer limit.

RulePROG6: Avoid type conversions.
The conversions are the responsibility of the programmer, not the compiler. Many errors originate from conversions.

RulePROG7: Declare *unsigned char* if using 8-bit ASCII.
The *char* type can not contain 8-bit ASCII. Many accents are no longer valid in 7-bit ASCII.

RulePROG8: Do not convert a *const* type to a *non-const* type.
This rule has no exceptions. The *const* attribute limits the modification of an object. By not converting a *const*type to *non-const* type, the developer can avoid certain side effects. For example, when researching an error, a developer may find that a method which receives a pointer of type *const* will not modify an object. The developer, therefore, may neglect to look at the body of the method. If the *const* attribute is modified to *non-const*, the developer will not verify the method's signature. Therefore, the developer will neglect to verify a function which has the ability to modify objects.

RulePRGO9: Open the file with the appropriate division rights and verify that the file has been opened.
If the file has to be opened by more than a single application, it is necessary to open the index with the corresponding rights.

RulePROG10: Each switch statement must have a default clause.
If necessary, use*assert (0)*, if this clause never has to be called.

RulePROG11: Avoid using very long *switch* type functions.

This prevents the optimization of the compiler and makes the code less comprehensible.

RulePROG12: All macro parameters must contain parentheses.
A macro call with an operator can modify the functionality if there are no parentheses.

RulePROG13: Never use memory which has just been freed.

Include

RulePROG14: The classes which are used with only one pointer or reference, do not have to be included in the header files (*.h).
Use the *forward* declarations in those cases.

Memory Management

RulePROG22: Always check the return of a *malloc* or of a *new* with NULL.
If necessary, before managing the error, add an *assert (pt!=NULL)* right after the *malloc*.

Portability

RulePROG23: Do not limit the program to a given memory model.
Always consider all of the models at the same time. 16-bit Windows is particularly demanding on the memory models.

Style

RulePROG24: All functions must have a prototype.

C Source

Debugging

RulePROG25: Always declare, for each object having a pointer, the constructor of the copy and the assignment operator.
If its functions never have to be called, declare them in *private*; otherwise, write them correctly.

Portability

RulePROG26: Avoid using a parameter received by reference in a constructor.

Style

RulePROG27: Reread the classes to ensure that the relations *is-an* and *uses-one* are correct.